Programare funcțională

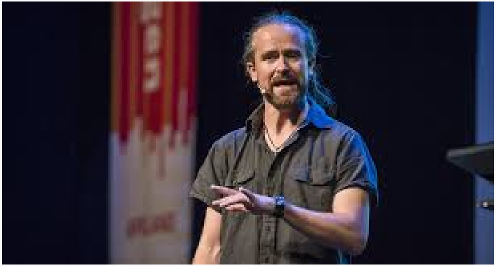
Introducere în programarea funcțională folosind Haskell
Curs Facultativ

Claudia Chiriță
Denisa Diaconescu

Departamentul de Informatică, FMI, UB

Corespondența Curry-Howard

Schimbați perspectiva



Roger Antonsen

Universitatea din Oslo

TED Talk: Math is the hidden secret to understanding the world

"... înțelegerea constă în abilitatea de a-ți schimba perspectiva"

https://www.ted.com/talks/roger_antonsen_math_is_the_hidden_secret_to_understanding_the_world

Un program simplu în Haskell

```haskell
data Point = Point Int Int

makePoint :: Int -> Int -> Point
makePoint x y = Point x y

getX :: Point -> Int
getX (Point x y) = x

getY :: Point -> Int
getY (Point x y) = y

origin :: Point
origin = makePoint 0 0
```

Un program simplu în Haskell

Hai să schimbăm perspectiva!

```haskell
data Point = Point Int Int

makePoint :: Int -> Int -> Point
makePoint x y = Point x y

getX :: Point -> Int
getX (Point x y) = x

getY :: Point -> Int
getY (Point x y) = y
```

$$\frac{x : Int \quad y : Int}{makePoint\ x\ y : Point}\ (Point_I)$$

$$\frac{p : Point}{getX\ p : Int}\ (Point_{E_1})$$

$$\frac{p : Point}{getY\ p : Int}\ (Point_{E_2})$$

Generalizare

$$\frac{x : Int \quad y : Int}{makePoint\ x\ y : Point}\ (Point_I) \qquad \frac{M : A \quad N : B}{\langle M, N\rangle : A \times B}\ (\times_I)$$

$$\frac{p : Point}{getX\ p : Int}\ (Point_{E_1}) \qquad \frac{M : A \times B}{fst\ M : A}\ (\times_{E_1})$$

$$\frac{p : Point}{getY\ p : Int}\ (Point_{E_2}) \qquad \frac{M : A \times B}{snd\ M : B}\ (\times_{E_2})$$

Alt exemplu simplu

```
f = (\x -> x * 3) :: Int -> Int
```

$$\frac{\{x : Int\} \vdash x * 3 : Int}{\lambda x.x * 3 : Int \to Int}\ (fun_I)$$

```
> f 5
15
```

$$\frac{f : Int \to Int \quad 5 : Int}{f\ 5 : Int}\ (fun_E)$$

Generalizare

$$\frac{\{x : Int\} \vdash x * 3 : Int}{\lambda x.x * 3 : Int \to Int}\ (fun_I)$$

$$\frac{\{x : A\} \vdash M : B}{\lambda x.M : A \to B}\ (\to_I)$$

$$\frac{f : Int \to Int \quad 5 : Int}{f\ 5 : Int}\ (fun_E)$$

$$\frac{M : A \to B \quad N : A}{M\,N : B}\ (\to_E)$$

Logica. Ce este adevăt și ce este fals?

Hai să schimbăm perspectiva iar!

Logica. Ce este adevăt și ce este fals?

Hai să schimbăm perspectiva iar!

Dacă afară este întuneric atunci,
dacă porcii zboară atunci este întuneric afară.

$$A = \text{afară este întuneric}$$
$$B = \text{porcii zboară}$$

$$A \supset (B \supset A)$$

Logica. Ce este adevăt și ce este fals?

Hai să schimbăm perspectiva iar!

Dacă afară este întuneric atunci,
dacă porcii zboară atunci este întuneric afară.

$$A = \text{afară este întuneric} \qquad A \supset (B \supset A)$$
$$B = \text{porcii zboară}$$

Este adevărată această afirmație? Da!

A	B	$B \supset A$	$A \supset (B \supset A)$
false	false	true	true
false	true	false	true
true	false	true	true
true	true	true	true

Semantica unei logici

Dăm valori variabilelor în mulțimea $\{0, 1\}$.

Definim o evaluare $e : V \rightarrow \{0, 1\}$.

Putem să o extindem o evaluare la formule:

$\wedge : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

$\supset : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$

A	B	$A \supset B$
0	0	1
0	1	1
1	0	0
1	1	1

Dacă pentru toate evaluările posibile, o formulă are valoarea 1, atunci spunem că este o tautologie.

Sintaxa unei logici

Dăm metode pentru a manipula simbolurile din logică (i.e., $\supset$, $\wedge$) pentru a stabili când o formulă este demonstrabilă/teoremă .

Corectitudine	=	sintaxa implică semantica
Completitudine	=	sintaxa și semantica coincid

Un sistem de deducție naturală

Reguli pentru a manevra fiecare conector logic (introducerea și eliminarea conectorilor).

$$\frac{\vdash A \qquad \vdash B}{\vdash A \wedge B}\ (\wedge_I) \qquad \frac{\vdash A \wedge B}{\vdash A}\ (\wedge_{E_1}) \qquad \frac{\vdash A \wedge B}{\vdash B}\ (\wedge_{E_2})$$

$$\frac{\{A\} \vdash B}{\vdash A \supset B}\ (\supset_I) \qquad \frac{\vdash A \supset B \qquad \vdash A}{\vdash B}\ (\supset_E)$$

Arată cunoscut?

Corespondența Curry-Howard

λ-calcul cu tipuri	Deducție naturală
$\dfrac{\vdash M : A \qquad \vdash N : B}{\vdash \langle M, N \rangle : A \times B}\ (\times_I)$	$\dfrac{\vdash A \qquad \vdash B}{\vdash A \wedge B}\ (\wedge_I)$
$\dfrac{\vdash M : A \times B}{\vdash fst\ M : A}\ (\times_{E_1})$	$\dfrac{\vdash A \wedge B}{\vdash A}\ (\wedge_{E_1})$
$\dfrac{\vdash M : A \times B}{\vdash snd\ M : B}\ (\times_{E_2})$	$\dfrac{\vdash A \wedge B}{\vdash B}\ (\wedge_{E_2})$
$\dfrac{\{x : A\} \vdash M : B}{\vdash \lambda x.M : A \to B}\ (\to_I)$	$\dfrac{\{A\} \vdash B}{\vdash A \supset B}\ (\supset_I)$
$\dfrac{\vdash M : A \to B \qquad \vdash N : A}{\vdash M\,N : B}\ (\to_E)$	$\dfrac{\vdash A \supset B \qquad \vdash A}{\vdash B}\ (\supset_E)$

Propositions are types! ♡

Să analizăm mai atent

λ-calcul cu tipuri	Deducție naturală
$\Gamma \vdash M : A$	$\Gamma \vdash A$

Faptul că există un termen de tip A *(inhabitation of type A)*
înseamnă că A este teoremă/are o demonstrație în logică! ♡

Să analizăm mai atent

λ-calcul cu tipuri

$$\frac{\{x : A\} \vdash x : A}{\vdash \lambda x.\, x : A \to A}\ (\to_I)$$

Deducție naturală

$$\frac{\{A\} \vdash A}{\vdash A \supset A}\ (\supset_I)$$

Să analizăm mai atent

λ-calcul cu tipuri

$$\frac{\{x : A\} \vdash x : A}{\vdash \lambda x.\, x : A \to A}\ (\to_I)$$

$$\cfrac{\cfrac{\overline{\{x : A, y : B\} \vdash x : A}}{\{x : A\} \vdash \lambda y.\, x : B \to A}\ (\to_I)}{\vdash \lambda x.\, (\lambda y.\, x) : A \to (B \to A)}\ (\to_I)$$

Deducție naturală

$$\frac{\{A\} \vdash A}{\vdash A \supset A}\ (\supset_I)$$

$$\cfrac{\cfrac{\overline{\{A, B\} \vdash A}}{\{A\} \vdash B \to A}\ (\supset_I)}{\vdash A \to (B \to A)}\ (\supset_I)$$

Să analizăm mai atent

λ-calcul cu tipuri

$$\frac{\{x : A\} \vdash x : A}{\vdash \lambda x.\, x : A \to A}\ (\to_I)$$

$$\frac{\dfrac{\overline{\{x : A, y : B\} \vdash x : A}}{\{x : A\} \vdash \lambda y.\, x : B \to A}\ (\to_I)}{\vdash \lambda x.\, (\lambda y.\, x) : A \to (B \to A)}\ (\to_I)$$

Deducție naturală

$$\frac{\{A\} \vdash A}{\vdash A \supset A}\ (\supset_I)$$

$$\frac{\dfrac{\overline{\{A, B\} \vdash A}}{\{A\} \vdash B \to A}\ (\supset_I)}{\vdash A \to (B \to A)}\ (\supset_I)$$

Proofs are Terms! ♡

Demonstrațiile sunt termeni!

Corespondența Curry-Howard

Teoria Tipurilor	Logică
tipuri	formule
termeni	demonstrații
inhabitation a tipului A	demonstrație a lui A

Corespondența Curry-Howard

Teoria Tipurilor	Logică
tipuri	formule
termeni	demonstrații
inhabitation a tipului A	demonstrație a lui A
tip produs	conjuncție
tip funcție	implicație

Corespondența Curry-Howard

Teoria Tipurilor	Logică
tipuri	formule
termeni	demonstrații
inhabitation a tipului A	demonstrație a lui A
tip produs	conjuncție
tip funcție	implicație
tip sumă	disjuncție
tipul void	false
tipul unit	true

Logica intuiționistă

- Logică constructivistă
- Bazată pe noțiunea de demonstrație
- Utilă deoarece demonstrațiile sunt executabile și produc exemple. Permite "extragererea" de programe demonstrate a fi corecte.
- Baza pentru *proof assistants* (e.g., Coq, Lean, Agda, Idris)
- Următoarele formule echivalente nu sunt demonstrabile în logica intuiționistă!
  - dubla negație: $\neg\neg\varphi \supset \varphi$
  - excluded middle: $\varphi \vee \neg\varphi$
  - legea lui Pierce: $((\varphi \supset \psi) \supset \varphi) \supset \varphi$
- Nu există semantică cu tabele de adevăr pentru logica intuiționistă! Semantici alternative (e.g., semantica de tip Kripke)

Corespondența Curry-Howard

Inițial, corespondența Curry-Howard a fost între

Calculul Church $\lambda \rightarrow$	Sistemul de deducție naturală al lui Gentzen pentru logica intuiționistă

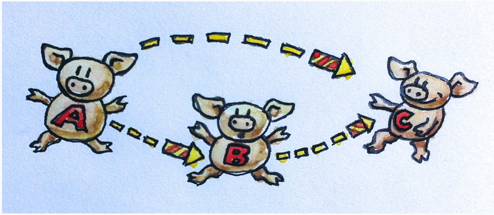
De ce?

- Este pur si simplu fascinant
- Nu gândiți logica și informatica ca domenii diferite.
- Gândind din perspective diferite ne poate ajuta să știm ce este posibil/imposibil.
- Teoria tipurilor nu ar trebui să fie o adunătură *ad hoc* de reguli!

Intro în Teoria Categoriilor

O categorie

- A category is an embarrassingly simple concept.
  Bartosz Milewski, *Category Theory for Programmers*
- Categorie = obiecte + săgeți
- Ingredient cheie: compunerea de săgeți



credits: Bartosz Milewski

Categorie

O categorie $\mathbb{C}$ constă în

- Obiecte: notate $A, B, C, \ldots$. Notăm cu $|\mathbb{C}|$ obiectele lui $\mathbb{C}$
- Săgeți: pentru orice obiecte A și B, există o mulțime de săgeți $\mathbb{C}(A, B)$
  - notăm $f \in \mathbb{C}(A, B)$ cu $f : A \to B$ sau $A \xrightarrow{f} B$
- Compunere: pentru orice săgeți $f : A \to B$ și $g : B \to C$ există o săgeată $f; g : A \to C$

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[dr, dashed, "f;g"'] & B \arrow[d, "g"] \\ & C \end{tikzcd}$$

- Identitate: pentru orice obiect A există o săgeată $id_A : A \to A$
- Axiome: pentru orice săgeți $f : A \to B$, $g : B \to C$, și $h : C \to D$, avem

$$f; (g; h) = (f; g); h \qquad f; id_B = f = id_A; f$$

Exemplu - categoria de mulțimi

Categoria $\mathbb{S}$et are

- Obiecte: mulțimi
- Săgeți: funcții
- Compunere: compunerea de funcții
- Identitate: pentru orice mulțime A, funcția identitate $id_A : A \to A$, $id_A(a) = a$
- Axiome: ✓

Monoizi

Un monoid **M** este o structură $\langle M, +, e \rangle$ astfel încât

- M este o mulțime
- $+ : M \times M \to M$ este asociativă
  (adică $(a + b) + c = a + (b + c)$ pentru orice $a, b, c \in M$)
- $e \in M$ este identitate pentru $+$
  (adică $e + a = a + e = a$ pentru orice $a \in M$)

Monoizii sunt un concept extrem de puternic:

- Stau în spatele aritmeticii de bază
  - și adunarea, și înmulțirea formează un monoid
- Sunt prezenți peste tot în programare
  - șiruri de caractere, liste ...

Exemplu - categoria de monoizi

Categoria $\mathbb{M}$on are

- Obiecte: monoizi
- Săgeți: morfisme de monoizi
  (adică funcții care nu "strică" operația de monoid)
- Compunerea: compunerea de morfisme de monoizi
- Identitatea: pentru orice obiect **M**, $id_{\mathbf{M}} : M \to M$, $id_M(m) = m$
- Axiome: ✓

Exemplu - un monoid ca o categorie

Orice monoid $\mathbf{M} = \langle M, +, e \rangle$ este o categorie cu

- Obiecte: un singur obiect $\heartsuit$
- Săgeți: elementele mulțimii M (adică $\mathbf{M}(\heartsuit, \heartsuit) = M$)
- Compunerea: operația de monoid $+$
- Identitatea: identitatea monoidului e
- Axiome:

$$
\begin{array}{cc}
f; (g; h) = (f; g); h & f; id_B = f = id_A; f \\
a + (b + c) = (a + b) + c & a + e = a = e + a
\end{array}
$$

Exemplu - Categoria $\mathbb{H}$ask

- Obiectele: tipuri
- Săgețiile: funcții între tipuri

```
f :: a -> b
```

- Identități: funcția polimorfică **id**

```
id :: a -> a
```

- Compunere: funcția polimorfică `(.)`

```
(.) :: (b -> c) -> (a -> b) -> a -> c
```

Subcategorii ale lui $\mathbb{H}$ask date de tipuri parametrizate

- Obiecte: o clasă restrânsă de tipuri din $|\mathbb{H}\mathrm{ask}|$

  Exemplu: tipuri de forma `[a]`

- Săgeți: toate funcțiile din $\mathbb{H}$ask între tipurile obiecte

  Exemple:

```
concat :: [[a]] -> [a]
words :: [Char] -> [String]
reverse :: [a] -> [a]
```

Exemple

$\mathbb{L}$**iste**	obiecte: tipuri de forma [a]
$\mathbb{O}$**ptiuni**	obiecte: tipuri de forma Maybe a
$\mathbb{F}$**uncții de sursă t**	obiecte: tipuri de forma t -> a

De ce categorii?

(Des)compunerea este esența programării

- Am de rezolvat problema P
- O descompun în subproblemele $P_1, \ldots P_n$
- Rezolv problemele $P_1, \ldots P_n$ cu programele $p_1, \ldots p_n$
  - Eventual aplicând recursiv procedura de față
- Compun rezolvările $p_1, \ldots p_n$ într-o rezolvare p pentru problema inițială

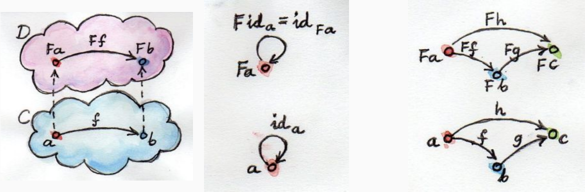
Categoriile rezolvă problema compunerii

- Ne forțează să abstractizăm datele
- Se poate acționa asupra datelor doar prin săgeți
- Forțează un stil de compunere independent de structura obiectelor

Functori

Date fiind două categorii $\mathbb{C}$ și $\mathbb{D}$, un functor $F : \mathbb{C} \to \mathbb{D}$ este dat de

- O funcție $F : |\mathbb{C}| \to |\mathbb{D}|$ de la obiectele lui $\mathbb{C}$ la cele ale lui $\mathbb{D}$
- Pentru orice $A, B \in |\mathbb{C}|$, o funcție $F : \mathbb{C}(A, B) \to \mathbb{D}(F(A), F(B))$ compatibilă cu identitățile și cu compunerea
  - $F(id_A) = id_{F(A)}$ pentru orice $A \in |\mathbb{C}|$
  - $F(f; g) = F(f); F(g)$ pentru orice $f : A \to B, g : B \to C$



Bartosz Milewski — Functors

Functori în Haskell

În Haskell o instanță **`Functor`** `f` este dată de

- Un tip `f a` pentru orice tip `a`
  (deci `f` trebuie să fie tip parametrizat)
- Pentru orice două tipuri `a` și `b`, o funcție

  ```
  fmap :: (a -> b) -> (f a -> f b)
  ```

- Compatibilă cu identitățile și cu compunerea

  ```
  fmap id == id
  fmap (g . h) == fmap g . fmap h
  ```

  pentru orice `h :: a -> b` și `g :: b -> c`

Corespondența Curry-Howard

Teoria Tipurilor	Logică
tipuri	formule
termeni	demonstrații
inhabitation a tipului A	demonstrație a lui A

Corespondența Curry-Howard

Teoria Tipurilor	Logică
tipuri	formule
termeni	demonstrații
inhabitation a tipului A	demonstrație a lui A
tip produs	conjuncție
tip funcție	implicație

Corespondența Curry-Howard

Teoria Tipurilor	Logică
tipuri	formule
termeni	demonstrații
inhabitation a tipului A	demonstrație a lui A
tip produs	conjuncție
tip funcție	implicație
tip sumă	disjuncție
tipul void	false
tipul unit	true

Să investigăm mai mult!

Obiect inițial și obiect final

Într-o categorie $\mathbb{C}$,

- un obiect T se numește terminal dacă pentru orice obiect A există o unică săgeată

  $$\tau_A : A \to T$$

  În $\mathbb{H}$ask, obiectul terminal este `()`.
  Pentru orice tip `a`, avem `unit :: a -> ()`.

- un obiect I se numește inițial dacă pentru orice obiect A există o unică săgeată

  $$\iota_A : I \to A$$

  În $\mathbb{H}$ask, obiectul inițial este **`Void`**.
  Pentru orice tip `a`, avem `absurd :: Void -> a`.

De ce obiect final?

Fie $\mathbb{C}$ o categorie cu obiect terminal T.
Avem următoarea interpretare:

- Formulele propoziționale sunt obiectele lui $\mathbb{C}$
- Constanta $\top$ (true) este obiectul terminal T
- O demonstrație a lui A este o săgeată $f : T \to A$
- O demonstrație a lui A din ipotezele B este o săgeată $f : B \to A$

Produse

Fie A și B obiecte într-o categorie $\mathbb{C}$.

Spunem că

$$A \xleftarrow{\pi_1} A \times B \xrightarrow{\pi_2} B$$

este produs al lui A și B dacă pentru orice

$$A \xleftarrow{f} C \xrightarrow{g} B$$

există o unică săgeată

$$\langle f, g \rangle : C \to A \times B$$

astfel încât

$$\langle f, g \rangle ; \pi_1 = f \qquad \langle f, g \rangle ; \pi_2 = g$$

Produse

$$\begin{tikzcd}
A & A \times B \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & B \\
& C \arrow[ul, "f"] \arrow[u, dashed, "\langle f,g \rangle"] \arrow[ur, "g"'] &
\end{tikzcd}$$

Produse

Fie $\mathbb{C}$ o categorie cu obiect terminal T și produse.

Fie A și B două obiecte în **C**.

$$
\begin{tikzcd}
A & A \times B \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & B \\
& T \arrow[ul, "f"] \arrow[u, dashed, "\langle f,g \rangle"] \arrow[ur, "g"'] &
\end{tikzcd}
$$

$$\frac{f : T \to A \quad g : T \to B}{\langle f, g \rangle : T \to A \times B}$$

$$\frac{\langle f, g \rangle : T \to A \times B}{\langle f, g \rangle \,;\, \pi_1 : T \to A}$$

$$\frac{\langle f, g \rangle : T \to A \times B}{\langle f, g \rangle \,;\, \pi_2 : T \to B}$$

Produse

Fie $\mathbb{C}$ o categorie cu obiect terminal T și produse.

Fie A și B două obiecte în **C**.

$$
\begin{tikzcd}
A & A \times B \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & B \\
& T \arrow[ul, "f"] \arrow[u, dashed, "\langle f,g \rangle"] \arrow[ur, "g"'] &
\end{tikzcd}
$$

$$\frac{f : T \to A \quad g : T \to B}{\langle f, g \rangle : T \to A \times B}$$

$$\frac{\langle f, g \rangle : T \to A \times B}{\langle f, g \rangle ; \pi_1 : T \to A}$$

$$\frac{\langle f, g \rangle : T \to A \times B}{\langle f, g \rangle ; \pi_2 : T \to B}$$

Vă aduce aminte de ceva?

Trei perspective

λ-calcul cu tipuri	Deducție naturală	O Categorie*
$\dfrac{a : A \quad b : B}{\langle a, b\rangle : A \times B}\ (\times_I)$	$\dfrac{A \quad B}{A \& B}\ (\&_I)$	$\dfrac{f : T \to A \quad g : T \to B}{\langle f, g\rangle : T \to A \times B}$
$\dfrac{p : A \times B}{\mathit{fst}\ p : A}\ (\times_{E_1})$	$\dfrac{A \& B}{A}\ (\&_{E_1})$	$\dfrac{\langle f, g\rangle : T \to A \times B}{\langle f, g\rangle ; \pi_1 : T \to A}$
$\dfrac{p : A \times B}{\mathit{snd}\ p : B}\ (\times_{E_2})$	$\dfrac{A \& B}{B}\ (\&_{E_2})$	$\dfrac{\langle f, g\rangle : T \to A \times B}{\langle f, g\rangle ; \pi_2 : T \to B}$

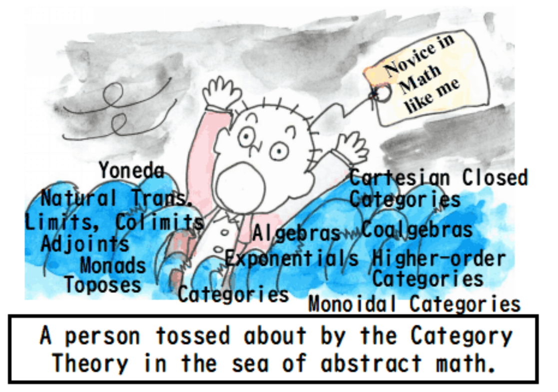
** O categorie cu obiect terminal T și produse*

Corespondența Curry-Howard-Lambek

Teoria tipurilor	Logică	Teoria Categoriilor
tipuri	formule	obiecte
termeni	demonstrații	săgeți
inhabitation a tipului A	demonstrație a lui A	săgeată $f : T \to A$
tip funcție	implicație	?
tip produs	conjuncție	produs
tip sumă	disjuncție	coprodus
tipul void	fals	obiect inițial
tipul unit	true	obiect terminal

Categorii Cartezian Închise

O categorie cu obiect terminal, produse și exponențiali se numește o **categorie cartezian închisă**. *Cartesian Closed Category (CCC)*



Pe săptămâna viitoare!